ANL/MCS-TM-289

Support Vector Machine Classifiers for Large Data Sets

prepared by
Mathematics and Computer Science Division
Argonne National Laboratory

ANL/MCS-TM-289

Support Vector Machine Classifiers for Large Data Sets

by
E.M. Gertz* and J.D. Griffin**
Mathematics and Computer Science Division, Argonne National Laboratory
Technical Memorandum ANL/MCS-TM-289

* Computer Sciences Department, University of Wisconsin, Madison, WI 53706. E-mail: emgertz@mac.com.
** Givens Associate 2001, Mathematics and Computer Science Division, Argonne National Laboratory
Computational Sciences and Mathematical Research Division, Sandia National Laboratories,
Livermore, CA 94551-9217. E-mail: jgriffi@sandia.gov.

October 2005

While the authors were affiliated with the Mathematics and Computer Science Division, this work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. This work was also supported by National Science Foundation grants ACI-0082100 and DMS-0208449.

Contents

Support Vector Machine Classifiers for Large Data Sets

by

E. Michael Gertz and Joshua D. Griffin

Abstract

This report concerns the generation of support vector machine classifiers for solving the pattern recognition problem in machine learning. Several methods are proposed based on interior-point methods for convex quadratic programming. Software implementations are developed by adapting the object-oriented packaging OOQP to the problem structure and by using the software package PETSc to perform time-intensive computations in a distributed setting. Linear systems arising from classification problems with moderately large numbers of features are solved by using two techniques—one a parallel direct solver, the other a Krylov-subspace method incorporating novel preconditioning strategies. Numerical results are provided, and computational experience is discussed.

1 Introduction

Researchers have expressed considerable interest in the use of support vector machine (SVM) classifiers in pattern recognition problems (see Burges [5]; Cristianini and Shawe-Taylor [6]; and Vapnik [28].) The problem of generating an SVM classifier can be reduced to one of solving a highly structured convex quadratic program. This quadratic program can be very large, and one must exploit the structure of the problem to solve efficiently.

A number of methods for solving the SVM subproblem have been developed. Active-set methods, such as those studied by Osuna et al. [20], Joachims [16], Platt [23, 22], and Keerthi et al. [26] are very popular. Other methods have also been proposed, such as the Lagrangian methods of Mangasarian and Musicant [18] and the semismooth methods of Ferris and Munson [8].

Recently, Ferris and Munson [7] have shown how to efficiently solve large problems, with millions of observations, using a primal-dual interior-point algorithm and specialized linear algebra. They vastly reduce the size of the linear systems to be solved by using the Sherman-Morrison-Woodbury (SMW) formula (see [12]), which is equivalent to using the Schur complement arising in a block row-reduction of the linear system. In this paper, we take a similar approach in formulating the SVM problem. Our goal is to reduce the overall cost of generating an SVM classifier by reducing the cost of applying the SMW formula, which is by far the most expensive operation in the algorithm. We explore two techniques. The first exploits the natural parallelism of the SMW formula and uses several processors to solve the problem. The second is based on the observation that the SMW formula involves a positive-definite matrix with a natural preconditioner. This suggests the application of a matrix-free Krylov-space iterative solver to the associated linear system.

In Section 2, we briefly review the theory of support vector machines. The use of interior-point methods to generate SVM classifiers is considered in Section 3. In Section 4 we describe how classifiers may be generated efficiently in a parallel environment. The proposed matrix-free iterative method is outlined in Section 5. Numerical experiments with an implementation using the object-oriented QP solver OOQP [11, 10] and the parallel linear algebra library PETSc [1, 2, 3] are described in Section 6.

2 Support Vector Machines

A learning machine finds a mapping, known as a classifier, between a population of objects and a set of labels. For the pattern recognition problem, the labels are "yes" and "no," which we represent here as 1 and -1. A support vector machine is a specific type of learning machine for the pattern recognition problem. The simplest SVM generates a linear classifier—an affine function $x \mapsto w^T x + \beta$ that is used to define the classifier

$$x \mapsto \begin{cases} 1 & \text{if } w^T x + \beta \geq 0; \\ -1 & \text{otherwise.} \end{cases}$$

Classifiers are created by determining appropriate values of w and β by observing the features of a training set, a subset of the population that has a known classification. Let n denote the number of observations in the training set. Let m be the number of features in each observation vector x_i, and let $d_i \in \{-1, 1\}$ indicate its classification. Let X denote the $n \times m$ matrix whose rows are the observations x_i; in other words, $X^T = (x_i \cdots x_n)$. Similarly, let D denote the $n \times n$ diagonal matrix $\mathrm{diag}(d)$. Then, a linear SVM classifier may be created by finding w and β that solve the minimization problem

$$\begin{array}{lrcl} \text{minimize} & \|w\|_2^2 + \tau e^T z & & \\ \text{subject to} & D(Xw - \beta e) & \geq & e - z \\ & z & \geq & 0, \end{array} \tag{1}$$

where e is a vector of all ones, z is a vector of appropriate size, and τ is a positive constant that is a parameter of the problem.

This formulation may be motivated by regarding $e^T z$ as a measure of the misclassification of the training set by the generated classifier. The term $\tau e^T z$ is known as an ℓ_1 penalty function in the theory of constrained optimization. A well-known property of the ℓ_1 penalty functions is that if there are values of w and β that separate the training data correctly, then these values will be the solution of the optimization problem for all sufficiently large τ (see, e.g., Fletcher [9]). It is easily shown that the distance between the two hyperplanes $x_i^T w + \beta = 1$ and $x_i^T w + \beta = -1$ is given by $2/\|w\|_2$. Thus the objective in the optimization problem (1) can be seen as a balance between trying to minimize the empirical misclassification error and trying to maximize the separation margin. (See Vapnik [28] for a discussion of the composite objective and of why a larger separation margin may improve the generalization capability of the classifier.)

The dual of the problem (1) is

$$\begin{array}{ll} \text{minimize} & -e^T v + v^T D X X^T D v \\ \text{subject to} & e^T D v = 0, \;\; 0 \leq v \leq \tau e. \end{array} \tag{2}$$

For the primal-dual methods described in Section 4, there is little difference between the primal (1) and dual (2) formulations. It is not hard to see that by making r_w identically zero in (5) and eliminating Δw from the system, one may obtain a primal-dual iteration on the dual problem (2). However, the constraints of the dual problem are mainly simple bounds, a fact that has been used to great effect in a number of algorithms, notably the chunking algorithms introduced by Osuna et al. [20].

The dual formulation has also been used to generalize the classical linear problem (1). This generalization involves replacing the product DXX^TD in (2) by a matrix Q such that $q_{ij} = d_i\mathcal{K}(x_i, x_j)d_j$, where $\mathcal{K}$ is a given kernel function $\mathcal{K} : \Re^n \times \Re^n \mapsto \Re$. This yields a problem of the form

$$\begin{array}{ll} \text{minimize} & -e^Tv + v^TQv \\ \text{subject to} & e^TDv = 0, \;\; 0 \le v \le \tau e. \end{array} \tag{3}$$

The $n \times n$ matrix Q is large and typically dense, making it inefficient to apply a primal-dual iteration naively to (3). Under suitable conditions, however, the use of a kernel function is equivalent to defining a transformation $\Phi(x)$ that maps the data into a larger, possibly infinite-dimensional feature space and finding a separating hyperplane in this space (see Burges [5] or Cristianini and Shawe-Taylor [6] for details). For some kernels, particularly polynomial kernels, the mapping $\Phi(x)$ is not hard to define. Recently Smola and Schölkopf [25] show how to use an incomplete Cholesky factorization to reduce the rank of Q, thereby implicitly providing an appropriate approximate $\Phi(x)$ that may be used in a primal-dual method.

3 Interior-Point Methods

The problem (1) has a convex quadratic objective and only linear constraints. A general class of methods that have proven effective in solving such a problem is interior-point methods. For a discussion of such methods, see Wright [29]. We focus on implementations based on Mehrotra's predictor-corrector (MPC) method [19] and Mehrotra's method with Gondzio's second-order correctors [13].

As a general rule, primal-dual interior point methods such as MPC operate by repeatedly solving Newton-like systems based on perturbations of the optimality conditions of the problem. For the SVM problem (1), these optimality conditions are

$$2w - Y^Tv = r_w = 0 \tag{4a}$$
$$d^Tv = \rho_\beta = 0 \tag{4b}$$
$$\tau e - v - u = r_z = 0 \tag{4c}$$
$$Yw - \beta d + z - e - s = r_s = 0 \tag{4d}$$
$$ZUe = 0 \tag{4e}$$
$$SVe = 0 \tag{4f}$$
$$s,\; u,\; v,\; z \ge 0, \tag{4g}$$

where $Y = DX$ and S, U, V, and Z are diagonal matrices whose diagonals are the elements of the correspondingly named vector. In (4b) we use ρ_β to denote the residual, rather than r_β, to emphasize that this quantity is a scalar. The Newton system for the equations (4a)–(4f) is

$$2\Delta w - Y^T\Delta v = -r_w \tag{5a}$$
$$d^T\Delta v = -\rho_\beta \tag{5b}$$
$$-\Delta v - \Delta u = -r_z \tag{5c}$$
$$Y\Delta w - d\Delta\beta + \Delta z - \Delta s = -r_s \tag{5d}$$
$$Z\Delta u + U\Delta z = -r_u \tag{5e}$$
$$S\Delta v + V\Delta s = -r_v, \tag{5f}$$

where $r_u = Zu$ and $r_v = Sv$. The MPC method solves systems with the same matrices, but for which the residuals r_u and r_v have been perturbed from their values in the Newton system.

The matrix of this system is large but sparse and highly structured. We use reductions similar to those described in Ferris and Munson [7] to reduce the system to a smaller dense system that may be efficiently solved by means of a Cholesky factorization. First, we eliminate the slack variables u and s from the system. Combining (5c) with (5e), and (5d) with (5f), we obtain the system

$$-\Delta v + Z^{-1}U\Delta z = -\hat{r}_z \tag{6a}$$

$$Y\Delta w - d\Delta\beta + \Delta z + V^{-1}S\Delta v = -\hat{r}_s, \tag{6b}$$

where the residuals are defined to be $\hat{r}_z = r_z + Z^{-1}r_u$ and $\hat{r}_s = r_s + V^{-1}r_v$. We may eliminate Δz from this system to obtain

$$Y\Delta w - d\Delta\beta + \Omega\Delta v = -r_\Omega,$$

where we define

$$\Omega = V^{-1}S + U^{-1}Z \tag{7}$$

and $r_\Omega = \hat{r}_s - U^{-1}Z\hat{r}_z$.

In matrix form, the remaining equations are

$$\begin{pmatrix} 2I & 0 & -Y^T \\ 0 & 0 & d^T \\ Y & -d & \Omega \end{pmatrix} \begin{pmatrix} \Delta w \\ \Delta\beta \\ \Delta v \end{pmatrix} = -\begin{pmatrix} r_w \\ \rho_\beta \\ r_\Omega \end{pmatrix}.$$

Simple row eliminations yield the block-triangular system

$$\begin{pmatrix} 2I + Y^T\Omega^{-1}Y & -Y^T\Omega^{-1}d & 0 \\ -d^T\Omega^{-1}Y & d^T\Omega^{-1}d & 0 \\ Y & -d & \Omega \end{pmatrix} \begin{pmatrix} \Delta w \\ \Delta\beta \\ \Delta v \end{pmatrix} = -\begin{pmatrix} r_w + Y^T\Omega^{-1}r_\Omega \\ \rho_\beta - d^T\Omega^{-1}r_\Omega \\ r_\Omega \end{pmatrix}.$$

A final row-reduction may be used to solve for Δw and $\Delta\beta$. Let us introduce the notation

$$\hat{r}_w = r_w + Y^T\Omega^{-1}r_\Omega \tag{8a}$$

$$\hat{\rho}_\beta = \rho_\beta - d^T\Omega^{-1}r_\Omega \tag{8b}$$

$$y_d = Y^T\Omega^{-1}d \tag{8c}$$

$$\sigma = d^T\Omega^{-1}d. \tag{8d}$$

The scalar σ is nonzero because Ω is positive definite and d is nonzero. Then Δw and $\Delta\beta$ may be found from the reduced system

$$(2I + Y^T\Omega^{-1}Y - \frac{1}{\sigma}y_d y_d^T)\Delta w = -(\hat{r}_w + \frac{1}{\sigma}\hat{\rho}_\beta y_d) \tag{9a}$$

$$\Delta\beta = \frac{1}{\sigma}(-\hat{\rho}_\beta + y_d^T\Delta w). \tag{9b}$$

The value of Δv may be obtained by the forward-substitution

$$\Delta v = -\Omega^{-1}(r_\Omega + Y\Delta w - d\Delta\beta). \tag{10}$$

Equation (6a) may then be used to compute $\Delta z = -U^{-1}Z(\hat{r}_z - \Delta v)$. The values of Δu and Δs may be obtained through the relations $\Delta u = -Z^{-1}(r_u + U\Delta z)$, and $\Delta s = -V^{-1}(r_v + S\Delta v)$, which are derived from (5e) and (5f), respectively.

Numerical experience has shown that the cost of solving this system is typically dominated by the cost of forming the matrix

$$M = 2I + Y^T\Omega^{-1}Y - \frac{1}{\sigma}y_d y_d^T, \tag{11}$$

which is the coefficient matrix for (9a). In particular, $O(nm^2/2)$ multiplications are needed to compute M explicitly. The next most significant operations, in terms of multiplications, are the $O(nm)$ multiplications needed to compute products with Y^T and Y when computing the residuals of the optimality conditions (4), in computing $\hat{r}_w$ and y_d in the row-reduction (8), and in computing Δv through (10). If M is computed explicitly, it requires $O(m^3/6)$ multiplications to perform a Cholesky factorization of M. If m is much smaller than n, as is typical in many applications, the cost of this factorization is minor.

We propose two techniques to reduce the time required to formulate and solve (9a). The first, described in Section 4, is to exploit the natural parallelism available in the formulation of M by bringing more computing hardware to bear on the problem. The second, described in Section 5, is to avoid the computation of M altogether and use an iterative method to solve (9a). This approach is suggested by the positive definiteness of M (see Section 5 for a proof).

4 A Parallel Direct Solver

The operations of the MPC algorithm may be implemented efficiently in a multiprocessor environment. The primal-dual interior-point algorithm must solve several perturbed Newton-systems, and all processors must perform calculations based on the computed steps. Thus, the algorithm has inherent sequence points—points at which all processors must wait for the completion of certain operations. However, the number of Newton-like systems that must be solved is typically modest, while the cost of solving each system dominates the computational cost of the algorithm. As we noted above, the most expensive operation is formulating and solving the system (9a). This system can be solved efficiently in a parallel environment.

We rewrite (9a) as $M\Delta w = b$, where $b = -(\hat{r}_w + \hat{\rho}_\beta y_d/\sigma)$, and M is defined in (11) but repeated here for convenience:

$$M = 2I + Y^T\Omega^{-1}Y - \frac{1}{\sigma} y_d y_d^T. \tag{12}$$

The computation of the vector b is straightforward to implement in a multiprocessor environment. Many of the operations require no communication because they involve multiplications by diagonal matrices or their inverses. The operations that most impact parallel efficiency are the multiplications by Y^T needed for the computation of $r_w = 2w - Y^T v$ in (4) and $\hat{r}_w = r_w + Y^T\Omega^{-1}r_\Omega$ and $y_d = Y^T\Omega^{-1}d$ in (8). However, the cost of these operations is insignificant compared to the cost of computing M.

Let ℓ denote the number of processors. We partition Y and the diagonal matrix Ω as follows:

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_\ell \end{pmatrix}, \quad \Omega^{-1} = \begin{pmatrix} \Omega_1^{-1} & & & \\ & \Omega_2^{-1} & & \\ & & \ddots & \\ & & & \Omega_\ell^{-1} \end{pmatrix}, \tag{13}$$

where Y_i and Ω_i are stored on processor i. With this notation

$$Y^T\Omega^{-1}Y = Y_1^T\Omega_1^{-1}Y_1 + Y_2^T\Omega_2^{-1}Y_2 + \cdots + Y_\ell^T\Omega_\ell^{-1}Y_\ell. \tag{14}$$

The outer product $Y_i^T\Omega_i^{-1}Y_i$ may be computed entirely on processor i with local data. We may use these products to compute M as follows. For $i = 2, \ldots, \ell$ let $M_i = Y_i^T\Omega_i^{-1}Y_i$, with $M_1 = 2I + Y_1^T\Omega_1^{-1}Y_1 - y_d y_d^T/\sigma$. Then M may be represented by

$$M = M_1 + M_2 + \cdots + M_\ell, \tag{15}$$

where M_i is stored entirely on processor i. The computation of M_1 requires additional communication, namely, the distribution of the elements of y_d to the first processor, but this cost is relatively modest.

Once M is computed as a sum of matrices stored across processors, it must be used to solve equation (9a). The approach we have taken is to explicitly compute M, reducing the sum (15) onto a single processor, processor number one. We then perform a Cholesky factorization and solve the system (9a) on this processor. The results are then distributed to the remaining processors. Our motivation for taking this sequential approach is that M is a $m \times m$ matrix, where m is typically relatively small. The Cholesky factorization may be implemented efficiently, and so performing this operation sequentially is not costly and will likely outperform a parallel implementation for small values of m. However, it may be advantageous to use a parallel Cholesky solver in a shared-memory environment (see, e.g., Golub and Van Loan [12]).

The cost of computing the sum (15) is not insignificant, but is modest. Each M_i is symmetric, and only $m(m+1)/2$ elements of M_i are significant. The actual cost of the reduction depends on the means of communication and the parallel library being used but may be as good as $O(\log \ell)$ communications of $m(m+1)/2$ elements.

Once Δw has been computed, it is necessary to compute the values: $\Delta\beta$, Δs, Δu, Δv, Δw, and Δz. The scalar $\Delta\beta$ may be computed without communication using (9b) because y_d and Δw are both available on processor one. The computation of Δv is the only step that requires communication because the multiplication by Y in (10).

5 Preconditioned Krylov-Space Methods

As discussed in Section (3), the cost of computing the matrix

$$M = 2I + Y^T \Omega^{-1} Y - \frac{1}{\sigma} y_d y_d^T \tag{16}$$

tends to dominate the overall computing time. Recall that the matrix M, first introduced in equation (11), is the coefficient matrix of the linear system (9a). An alternative to computing M is to apply a matrix-free iterative method to solving the system (9a). These iterative methods, also known as Krylov-space methods, do not require that M be computed explicitly; they require only a mechanism for computing matrix-vector products of the form Mx.

A total of $O(2nm+n)$ multiplications are needed to compute the matrix-vector product Mx if the computation is done in the straightforward manner. On the other hand, $O(nm^2/2)$ multiplications are required to compute M explicitly. Thus, in order to gain any advantage from using an iterative method, the system must be solved in fewer than $m/4$ iterations. For small values of m, there is little hope that this will be the case, but if m is moderately large, then an iterative strategy can be effective. However, the effectiveness and efficiency of an iterative method strongly depends on the availability of an adequate preconditioner.

5.1 The SVM Preconditioner

The Krylov-space method that we use is the preconditioned linear conjugate-gradient (PCG) method. We do not discuss the PCG method in detail, but rather refer the reader to Golub and Van Loan [12] and the references therein. A technique for defining a suitable preconditioner is to find a matrix that in some sense approximates the matrix of the linear system to be solved, yet is inexpensive to factor using a direct method. In this section we describe a suitable preconditioner for the linear system (11). In Section 5.2, we investigate properties of this preconditioner that make it a good choice for this system.

A predictor-corrector interior method solves two linear systems per iteration with the same matrix but different right-hand sides. Thus, it might be supposed that a direct factorization of M has some advantage over a Krylov-space method because the factorization need only be computed once. To some degree, this supposition is true, but the apparent advantage of direct methods is partially negated by three facts. First, the preconditioning matrix used by the iterative method need be computed only once per iteration; thus the iterative method derives some benefit from solving multiple systems with the same matrix. Second, it is not necessary to solve both the predictor and corrector steps to the same accuracy. Third, the right-hand sides of the systems differ but are related. Therefore the computed predictor step can be profitably used as an initial guess for the combined predictor-corrector step.

The product $Y^T\Omega^{-1}Y$ may be written as the sum of outer products

$$Y^T\Omega^{-1}Y = \frac{1}{\omega_1}y_1y_1^T + \cdots + \frac{1}{\omega_n}y_ny_n^T. \tag{17}$$

Recall from (7) that

$$\omega_i = s_i/v_i + z_i/u_i.$$

The complementarity conditions (4e) and (4f) require that at the solution $v_is_i = 0$ and $u_iz_i = 0$ for all $i \in 1, \ldots, n$. A common regularity assumption, known as strict complementarity, is that $v_i + s_i > 0$ and $u_i + z_i > 0$ at the solution. For any i for which strict complementarity holds, the limit of ω_i as the iterations progress is necessarily zero or infinity.

It follows that in the later iterations of a primal-dual algorithm, the terms in the sum (17) have widely differing scales. A natural approximation to $Y^T\Omega^{-1}Y$ is obtained by either omitting terms in the sum (17) that are small or by replacing these small terms by a matrix containing only their diagonal elements. We have found that the strategy of retaining the diagonal elements is more effective.

Let $\mathcal{A}$ be the set of indices for which the terms in the sum (17) are large in a sense that we make precise below. An appropriate preconditioner for M, which we henceforth refer to as the SVM preconditioner, is then

$$P_{\mathcal{A}} = 2I - \frac{1}{\tilde{\sigma}}\tilde{y}_d\tilde{y}_d^T + \sum_{i\in\mathcal{A}}\frac{1}{\omega_i}y_iy_i^T + \sum_{i\notin\mathcal{A}}\operatorname{diag}\Big(\frac{1}{\omega_i}y_iy_i^T\Big), \tag{18}$$

where $\tilde{y}_d = \sum_{i\in\mathcal{A}}(d_i/\omega_i)y_i$ and

$$\tilde{\sigma} = \begin{cases} \sum_{i\in\mathcal{A}}\omega_i^{-1} & \text{if } \mathcal{A} \text{ is nonempty;} \\ 1 & \text{otherwise.} \end{cases}$$

If chosen wisely, the size of $\mathcal{A}$ may be significantly smaller than the number of observations, allowing P to be formed at significantly less cost than it would take to form M.

The PCG method is well defined and convergent if both M and preconditioning matrix $P_{\mathcal{A}}$ are positive definite. As the following propositions show, this is always the case.

Proposition 1 *Let $\mathcal{I}$ be a nonempty subset of the integers from 1 to n. Let $\widetilde{\Omega}$ be the diagonal matrix whose diagonal elements are ω_i for $i \in \mathcal{I}$. Let $\tilde{d}$ and $\widetilde{Y}$ be defined similarly, with the $\tilde{d}$ defined to be the vector whose elements are d_i for $i \in \mathcal{I}$ and $\widetilde{Y}$ the matrix whose rows are y_i for $i \in \mathcal{I}$. Then the matrix*

$$Q = \widetilde{Y}^T \widetilde{\Omega}^{-1} \widetilde{Y} - \left(\tilde{d}^T \widetilde{\Omega}^{-1} \tilde{d}\right)^{-1} \left(\widetilde{Y}^T \widetilde{\Omega}^{-1} \tilde{d}\right) \left(\widetilde{Y}^T \widetilde{\Omega}^{-1} \tilde{d}\right)^T$$

is positive semidefinite.

Proof. Because $\widetilde{\Omega}^{-1}$ is positive definite, one may define the inner product

$$\langle x, y \rangle_\omega = x^T \widetilde{\Omega}^{-1} y$$

and associated inner-product norm $\|x\|_\omega = \sqrt{\langle x, x \rangle_\omega}$. For a vector v,

$$v^T Q v = \|\widetilde{Y} v\|_\omega^2 - \frac{\langle \widetilde{Y} v, \tilde{d} \rangle_\omega^2}{\|\tilde{d}\|_\omega^2}.$$

But by the Cauchy-Schwarz inequality $|\langle \widetilde{Y} v, \tilde{d} \rangle_\omega| \le \|\widetilde{Y} v\|_\omega \|\tilde{d}\|_\omega$. It immediately follows that Q is positive semidefinite. $\square$

Proposition 2 *For any index set $\mathcal{A}$, the matrix $P_\mathcal{A}$ defined by* (18) *is positive definite. Furthermore, the matrix M of* (12) *is positive definite.*

Proof. The preconditioner, $P_\mathcal{A}$, has the general form

$$P_\mathcal{A} = 2I + \sum_{i \notin \mathcal{A}} \operatorname{diag}\left(\frac{1}{\omega_i} y_i y_i^T\right) + Q.$$

If $\mathcal{A}$ is empty, then $Q = 0$. If $\mathcal{A}$ is nonempty, then Q satisfies the conditions of Proposition 1. In either case Q is positive semidefinite. For any vector v, the product vv^T is positive semidefinite and therefore has nonnegative diagonal elements. Thus $P_\mathcal{A}$ is the sum of $2I$ with several positive semidefinite matrices. It follows that $P_\mathcal{A}$ is positive definite.

If $\mathcal{A}$ contains all integers from 1 to n, then $P_\mathcal{A} = M$. Hence, it immediately follows that M is positive definite. $\square$

Next we develop a rule for choosing the set $\mathcal{A}$ at each iteration. Consider the quantity

$$\mu = (v^T s + u^T z)/(2n),$$

and observe that by definition $v_i s_i \le \mu$ and $u_i z_i \le \mu$ for all $i = 1, \ldots, n$. For a typical convergent primal-dual iteration, we have

$$v_i s_i \ge \rho\mu \text{ and } u_i z_i \ge \rho\mu \tag{19}$$

for some positive value ρ that is constant for all iterations. Thus those w_i that converge to zero typically converge at a rate proportional to μ. Similarly, those w_i that grow without bound typically grow at a rate proportional to $1/\mu$.

Based on the observations above, we choose $i \in \mathcal{A}$ if and only if

$$\frac{1}{\omega_i} y_i^T y_i \ge \gamma \min\left(1, \mu^{1/2}\right), \tag{20}$$

where the value γ is a parameter of the algorithm. Eventually the test will exclude all ω_i that are converging to zero at a rate proportional to μ.

One does not know *a priori* the most effective value for the parameter γ for a particular problem, although we have found that $\gamma = 100$ works well in practice. Sometimes, however, typically at an early primal-dual iteration, the PCG method will converge slowly or will diverge because of numerical error. In such cases, we use a heuristic to decrease the value of γ and include more indices in $\mathcal{A}$. We describe this heuristic in Section 5.3. We emphasize that γ never increases during the solution of a particular SVM problem.

5.2 Analysis of the SVM Preconditioner

For any positive-definite A, the worst-case convergence rate of the conjugate-gradient method is described by the inequality

$$\|x - x_k\| \leq 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k, \tag{21}$$

where $\kappa = \kappa_2(A) \triangleq \|A\|_2 \|A^{-1}\|_2$, is the condition-number of A and $\|v\|_A \triangleq \sqrt{v^T A v}$ for any vector v. (For a derivation of this bound, see, e.g., Golub and Van Loan [12].)

For the PCG method with the SVM preconditioner, the relevant condition number is $\kappa_2(P_{\mathcal{A}}^{-1} M)$. In this section, we show that the definition (18) of $P_{\mathcal{A}}$, together with the rule (20) for choosing indices to include in $\mathcal{A}$, implies that $\kappa_2(P_{\mathcal{A}}^{-1} M)$ converges to one as μ converges to zero. Thus, as the optimization progresses, $P_{\mathcal{A}}$ becomes an increasingly accurate preconditioner for M.

Note that both M and $P_{\mathcal{A}}$ have the form $2I + G$, where G denotes a positive-semidefinite matrix. This motivates the following proposition.

Proposition 3 *If $A = 2I + G$, where G is positive-semidefinite, then A is invertible, and $\|A^{-1}\|_2 \leq \frac{1}{2}$.*

Proof. For any vector v, it holds that

$$v^T A v = 2v^T v + v^T G v \geq 2\|v\|_2^2.$$

The Cauchy-Schwartz inequality gives $\|v\|_2 \|Av\|_2 \geq v^T A v$, which implies that $\|Av\|_2 \geq 2\|v\|_2$. This inequality establishes the nonsingularity of A because $Av = 0$ implies that $v = 0$.

If u is any vector such that $\|u\|_2 = 1$, then

$$1 = \|AA^{-1}u\|_2 \geq 2\|A^{-1}u\|_2.$$

But $\|A^{-1}\|_2 = \max_{\|u\|_2 = 1} \|A^{-1}u\|_2$, and so $\|A^{-1}\|_2 \leq \frac{1}{2}$. $\square$

The preceding proposition yields a bound on $\kappa_2(P_{\mathcal{A}}^{-1} M)$.

Proposition 4 *Let M and $P_{\mathcal{A}}$ be defined as in* (12) *and (18), respectively. If $N \triangleq M - P_{\mathcal{A}}$, then*

$$\kappa_2(P_{\mathcal{A}}^{-1} M) \leq \left(1 + \frac{1}{2}\|N\|\right)^2.$$

Proof. Note that $P_{\mathcal{A}}^{-1}M = P_{\mathcal{A}}^{-1}(P_{\mathcal{A}} + N) = I + P_{\mathcal{A}}^{-1}N$. Therefore, from Proposition (3) it follows that

$$\|P_{\mathcal{A}}^{-1}M\|_2 \leq 1 + \|P_{\mathcal{A}}^{-1}\|\|N\| \leq 1 + \frac{1}{2}\|N\|.$$

It follows from a similar argument that

$$\|(P_{\mathcal{A}}^{-1}M)^{-1}\|_2 = \|M^{-1}P_{\mathcal{A}}\|_2 \leq 1 + \frac{1}{2}\|N\|.$$

The result follows. $\square$

The SVM preconditioner is specifically designed to bound N.

Proposition 5 *Let $P_{\mathcal{A}}$ be defined by* (18)*, and let $\mathcal{A}$ be chosen by the rule* (20) *for some $\gamma \leq \hat{\gamma}$, where the upper bound $\hat{\gamma}$ is independent of the iteration number. Then there is a postive constant ξ such that, for any positive choice of the variables v, s, u, and z, it holds that $\|M - P_{\mathcal{A}}\|_2 \leq \xi\mu^{1/2}$.*

Proof. Consider the matrix difference

$$M - P_{\mathcal{A}} = \sum_{i \notin \mathcal{A}} \left(\frac{1}{\omega_i} y_i y_i^T - \operatorname{diag}\Big(\frac{1}{\omega_i} y_i y_i^T\Big) \right) - \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T + \frac{1}{\sigma} y_d y_d^T.$$

By rule (20), $y_i^T y_i / \omega_i < \hat{\gamma}\mu^{1/2}$ for $i \notin \mathcal{A}$. But then $\|y_i y_i^T\|_2/\omega_i \leq y_i^T y_i/\omega_i < \hat{\gamma}\mu^{1/2}$. Because $y_i y_i^T$ is a rank-one matrix, its two norm is equal to its Frobenius norm. Subtracting the diagonal elements from a matrix can only decrease the Frobenius norm, and so for $i \notin \mathcal{A}$,

$$\left\| \frac{1}{\omega_i} y_i y_i^T - \operatorname{diag}\Big(\frac{1}{\omega_i} y_i y_i^T\Big) \right\|_2 \leq \hat{\gamma}\mu^{1/2}.$$

The identities $y_d = \sum_{i=1\ldots m} (d_i/\omega_i) y_i$ and $y_d - \tilde{y}_d = \sum_{i \notin \mathcal{A}} (d_i/\omega_i) y_i$ imply that there are constants c_1, c_2, and c_3 depending only on the vectors $\{y_i\}$ such that

$$\|y_d - \tilde{y}_d\|_2 \leq c_1 \hat{\gamma} \mu^{1/2} \quad (22)$$

$$\|y_d - \tilde{y}_d\|_2 \leq c_2 \sigma \quad (23)$$

$$\|y_d\|_2 \leq c_3 \sigma. \quad (24)$$

If $\mathcal{A}$ is empty, then $\tilde{y}_d = 0$ and $\tilde{y}_d \tilde{y}_d^T/\tilde{\sigma} = 0$. Therefore, if $\mathcal{A}$ is empty, it follows that

$$\left\| \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T - \frac{1}{\sigma} y_d y_d^T \right\|_2 \leq c_1 c_3 \hat{\gamma} \mu^{1/2}.$$

Thus there is an ξ_1 for which $\|M - P_{\mathcal{A}}\|_2 \leq \xi_1 \mu^{1/2}$ whenever $\mathcal{A}$ is empty.

If $\mathcal{A}$ is nonempty, we define the necessarily nonnegative value $\Theta = (\sigma - \tilde{\sigma})/\tilde{\sigma}$. We may expand the product

$$\frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T = \frac{1}{\sigma}(1 + \Theta)\big(y_d - (y_d - \tilde{y}_d)\big)\big(y_d - (y_d - \tilde{y}_d)\big)^T,$$

gather terms, and apply norm inequalities to determine that

$$\begin{aligned} \left\| \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T - \frac{1}{\sigma} y_d y_d^T \right\|_2 \leq{}& \frac{\Theta}{\sigma}\|y_d\|_2^2 + 2\Big(\frac{1+\Theta}{\sigma}\Big)\|y_d - \tilde{y}_d\|_2 \|y\|_2 \\ &+ \Big(\frac{1+\Theta}{\sigma}\Big)\|y_d - \tilde{y}_d\|_2^2. \end{aligned} \quad (25)$$

We seek to establish an upper bound on Θ. When $\mathcal{A}$ is nonempty, then for whatever value of $\gamma \le \hat{\gamma}$ is chosen for that particular iteration, there is at least one i that satisifes rule (20). Hence,

$$\tilde{\sigma} = \sum_{i \in \mathcal{A}} \omega_i^{-1} \ge c_4 \gamma \min\left(1, \mu^{1/2}\right)$$

for a constant c_4 that depends only on the data vectors $\{y_i\}$. Similarly,

$$\sigma - \tilde{\sigma} = \sum_{i \notin \mathcal{A}} \omega_i^{-1} \le c_5 \gamma (n-1) \min\left(1, \mu^{1/2}\right),$$

for a constant c_5 depending only on the data. It follows that whenever $\mathcal{A}$ is nonempty,

$$\Theta = \frac{\sigma - \tilde{\sigma}}{\sigma}\left(1 + \frac{\sigma - \tilde{\sigma}}{\tilde{\sigma}}\right) \le \frac{\sigma - \tilde{\sigma}}{\sigma}\left(1 + (n-1)\frac{c_4}{c_5}\right). \tag{26}$$

We emphasize that this bound is independent of the choice of γ so long as $\mathcal{A}$ is nonempty. Thus, we may combine inequalites (22)–(26) to conclude that there is an ξ_2 so that $\|M - P_{\mathcal{A}}\|_2 \le \xi_2 \mu^{1/2}$ whenever $\mathcal{A}$ is nonempty. The result follows by taking ξ to be the larger of ξ_1 and ξ_2. $\square$

5.3 Termination Criteria for the PCG Method

The PCG method does not solve linear systems exactly, but only to some relative tolerance that is typically much greater than machine precision. We have found it advantageous to solve the early primal-dual systems to low accuracy and to increase the accuracy as iterations proceed.

Consider a single primal-dual iteration, and let $Mx = b_P$ denote the system (9a) with the right-hand side associated with the predictor step. Similarly, let $Mx = b_C$ denote the system associated with the corrector step. Let

$$\texttt{rtol} = \min(10^{-1}, 10^{-1}\mu).$$

The PCG algorithm is terminated when the predictor step, x_P, satisifes the condition

$$\|b_P - Mx_P\| \le \max\left(\texttt{rtol} \times \|b_P\|, 10^{-12}\right).$$

For the corrector step, x_C, we tighten the tolerance and impose the condition

$$\|b_C - Mx_C\| \le \max\left(10^{-2} \times \texttt{rtol} \times \|b_C\|, 10^{-12}\right).$$

We maintain a count of the cumulative number of PCG iterations used in solving both the predictor and corrector equations. While solving either system, if the cumulative number of iterations exceeds

$$i_{\max} = \max\left(m/8,\ 20\right),$$

then the value of γ is decreased, thereby increasing the size of the active set. The preconditioner is then updated before proceeding to the next PCG iteration.

The following rule is used to adjust the value of γ. For $j \ge 1$, let

$$k_j = \min\left(|\mathcal{A}| + jm/2,\ n\right).$$

Then, if d_j is the k_j^{th} largest value of $y_i^T y_i / \omega_i$ for $i = 1, \ldots, n$, define

$$\gamma_j = (1 - 10^{-8}) \frac{d_j}{\min\left(1, \mu^{1/2}\right)},$$

and let $\mathcal{A}_j$ be the set of indices chosen if $\gamma = \gamma_j$ in the rule (20). The intent of these definitions is to ensure that the size of $\mathcal{A}_j$ is at least k_j. For each $j \geq 1$ in turn, we use the indices in $\mathcal{A}_j$ to form the preconditioner, reset the cumulative iteration count to zero, and continue the PCG iteration for at most $i_{\max}$ additional iterations. Note that $\mathcal{A} \subset \mathcal{A}_1$ and that $\mathcal{A}_j \subset \mathcal{A}_{j+1}$ for $j \geq 1$. Therefore for each j, the preconditioner may be updated rather than being recalculated. Because the preconditioner is exact for sufficiently large j, there must be some γ_j for which both predictor and corrector systems converge. We choose this final γ_j to be γ for subsequent iterations of the optimization.

6 Numerical Results

Both the preconditioned conjugate-gradient and the parallel direct methods were implemented by using the object-oriented quadratic programming code OOQP. OOQP implements Mehrotra's predictor-corrector algorithm entirely in terms of an abstract representation of a convex quadratic program. We implemented the solvers described in this paper by providing concrete implementations of the necessary abstract routines and data structures, tailoring these implementations to handle the SVM subproblem efficiently.

For the solver that uses the preconditioned conjugate gradient method to solve linear systems, we use an algorithm, supplied by OOQP, that follows Mehrotra's algorithm, with minimal modifications necesary to handle quadratic, rather than linear, programs. For the parallel direct method, we use an algorithm that has been further modified to use additional corrector steps, as described by Gondzio [13]. The parallel direct solver can compute these additional corrector steps at little additional cost. Both these algorithms are described further in Gertz and Wright [11].

OOQP also uses an abstract representation of core linear algebra objects such as vectors, matrices, and linear solvers. We use the toolkit PETSc to provide implementations of these core objects and operations. PETSc is designed for parallel computation, so it is a natural choice for implementing the parallel direct solver. Moreover, PETSc implements a number of Krylov-space methods and in particular provides the PCG method. It was therefore convenient to use PETSc to implement both the SVM solvers described in this paper.

For all problems described below, we set

$$w = 0,\ \beta = 0,\ z = u = 2e, \ \text{ and } s = v = 2e,$$

to form an intial starting point. The problems were solved by using the penalty parameter $\tau = 1$. The QP solvers in OOQP are said to converge when

$$\mu = (v^T s + u^T z)/(2n) < 10^{-8} \text{ and } \|r\|_\infty < 10^{-8},$$

where

$$r = (r_w,\, \rho_\beta,\, r_z,\, r_s)^T,$$

with r_w, ρ_β, r_z, and r_s defined in (4).

To test the effectiveness of the parallel direct method, we used Chiba City, a scalable cluster at Argonne. Chiba City has 256 computing nodes, each consisting of a dual-CPU Pentium III 500 MHz system with 512 MB of RAM and 9 GB of local disk memory. For high-performance communication, 64-bit Myrnet is used. All computing nodes are connected via a switched fast internet. Communication between processors is accomplished by using the MPICH [14, 15] implementation of the standard Message Passing Interface (MPI) protocol.

Table 1 shows the performance of the parallel direct method for a varying number of processors on the same randomly generated problem with 400 features and 32,000 observations. We used random data in this case because the time to form and factor M depends only on the size of the data and not on the data itself. Column 2 reports the elapsed time required to solve the problem. For up to 16 processors, these times demonstrate excellent parallel efficiency. Column 3 reports the number of optimization iterations required to solve the problem, which is also the number of times M is formed and factored. As desired, the number of iterations does not increase as the number of processors is increased, evidence that the implementation is numerically stable.

Table 1: Numerical results for parallelized direct method on random data with 400 features and $32,000$ observations

Number of Processors	**Time**	**Iterations**
2	14 min 52 sec	19
4	8 min 5 sec	19
8	3 min 52 sec	19
16	2 min 4 sec	19
32	2 min 58 sec	19

To test the iterative solver, we used five problems taken from a repository of machine learning composed by Hettich, Blake, and Merz [24] at the University of California, Irvine:

Mushroom Database The mushroom database was provided by the Audobon Society Field Guide [24]. The database provides a list of mushrooms described in terms of physical characteristics and categorized as either poisonous or edible. It contains 8,124 data points each having 22 attributes.

Isolet Spoken Letter Recognition Database This database was created by Ron Cole and Mark Fanty [24]. Data was generated from 150 subjects who spoke each letter of the alphabet twice. Two data sets were provided, which we concatenated to create a single data set with 7,779 data points and 617 features.

Waveform Data Generator The waveform database arises from [4]. Several data sets are provided. We use the largest data set, waveform+noise, which contains 5,000 data points each having 40 attributes.

Letter Recognition Database The letter recognition database was created by David J. Slate [24]. The objective is to identify 26 capital letters in the English alphabet from rectangular pixel displays. The database consists of 20,000 data points each having 17 attributes.

Connect-4 Opening Database The connect-4 database "contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet and in which the next move is not forced." [24]. The set consists of 67,557 data points each having 42 categories.

The data points for all problems analyzed except isolet were projected to higher dimensions via the mapping $\phi : \mathbb{R}^m \to \mathbb{R}^{m_p}$, where $m_p = \binom{m+2}{2}$,

$$\phi(x) = \left(\phi_1(x)^T, \, \phi_2(x)^T, \, \phi_3(x)^T, \, 1\right)^T$$

and

$$\begin{aligned}
\phi_1(x) &= \left(x_1^2,\, x_2^2,\, \ldots,\, x_m^2\right), \\
\phi_2(x) &= \sqrt{2}\,(x_1x_2,\, \ldots,\, x_1x_m,\, x_2x_3,\, \ldots,\, x_2x_m,\, \ldots,\, x_{m-1}x_m), \\
\phi_3(x) &= \sqrt{2}\,(x_1,\, \ldots,\, x_m).
\end{aligned}$$

This construction is equivalent to the use of the polynomial kernel

$$\langle \phi(x), \phi(y) \rangle = (\langle x, y \rangle + 1)^2.$$

See [17, 21, 27] for a discussion of the use of polynomial kernels. The data for each problem was then normalized to have values lying in $[-1, 1]$, via the substitution

$$X_{ij} \leftarrow \frac{X_{ij}}{\max_{ij} |X_{ij}|}.$$

All problems were solved by using the penalty parameter $\tau = 1$ and $\gamma = 100$ as the initial value in (20).

The iterative solver was tested on one processor of a dual Xeon 3.6 GHz Intel processor CPU with 8 GB of RAM. We compare the performance of two similar methods for solving each SVM problem. The first method, which for the sake of brevity we call the *iterative solver*, uses the MPC algorithm and solves each instance of the linear system (12) using the PCG method and the SVM preconditioner. The other method, which we call the *serial direct solver*, also uses the MPC algorithm but solves (12) using a direct solver. The serial direct solver was implemented by simply setting $\gamma = 0$ in (20), which forces the preconditioner to be the matrix M itself and the PCG method to converge in one iteration.

Table 2 shows the time and number of MPC iterations used by the serial direct solver to solve each problem. We compare these data to the data of Table 3, which shows the time and number of MPC iterations used by the iterative solver. In every case, the iterative solver used less time than did the serial direct solver. Moreover, for every problem except mushroom, both solvers used the same number of MPC iterations; for mushroom the iterative solver used 22 iterations and the serial direct solver used 23. Thus for these problems, the lower relative accuracy of the PCG method compared to a direct method did not adversely impact the optimization solver.

Table 2: Numerical results for serial direct method on data sets from [24]

Name	**Hyperplane Dimension**	**Number of Data Points**	**Time (seconds)**	**MPC Itns.**
mushroom	276	8124	14.93	23
isolet	617	7797	178.9	25
waveform	861	5000	202.66	22
letter-recognition	153	20000	35.34	62
connect-4	946	67557	3825.36	26

The size of each problem is shown in Table 2. The second column of Table 3 shows the average number of vectors used to form the SVM preconditioner. In every case, this number is significantly smaller than the number of datapoints, which implies that on average it required fewer operations to form the SVM preconditioner than it would have required to form M. As we discussed in Section 5, one must be able to solve the linear system in fewer than $m/4$ iterations to derive any benefit from using the PCG method. The fifth column of Table 3 lists the average number of PCG iterations required for each MPC iteration. For this set of problems, the average number of PCG iterations is always significantly smaller than $m/4$.

Table 3: Numerical results for PCG method on data sets from [24]

Name	Avg. $\lvert\mathcal{A}\rvert$	Time (sec.)	MPC Itns.	Avg. PCG Itns.
mushroom	250	5.96	22	15.95
isolet	165	19.11	25	18.20
waveform	196	25.21	22	33.86
letter-recognition	605	20.49	62	13.90
connect-4	26110	1092.62	26	34.19

7 Conclusions

We have described a primal-dual interior-point method for determining SVM classifiers with respect to the binary classification problem. We have used both the MPC method and Mehrotra's method with Gondzio's second order correctors [13] for solving the outlying QP problem (1). The most time-consuming step in these optimization algorithms is obtaining a solution to (9a). Efficiently determining an approximate solution to (9a) has been the primary focus of this paper.

Two methods were described and implemented, with numerical results included. The parallel direct method appears to be robust and straightforward, yielding immediate and obvious gains. However, there is a limiting point, dependent on the dimensions of the problem, where adding more processors is no longer beneficial because of the increase in communication. The SVM iterative method was superior to the serial direct solver for all problems tested.

The effectiveness of both the parallel direct solver and the serial iterative solver hints that combining these approaches may lead to increased efficiency. In such an approach, $P_{\mathcal{A}}$ would be formed in parallel, as opposed M. In order to maximize the effectiveness of a parallel iterative method, however, necessary precautions would need be taken to ensure a proper distribution of data. Little would be gained from forming $P_{\mathcal{A}}$ in parallel if all vectors corresponding to $\mathcal{A}$ were to lie on a single processor. Robust and effective methods of balancing the cost of forming $P_{\mathcal{A}}$ across available processors are the subject of further research.

Acknowledgments

We thank Todd Munson for informative discussions about support vector machine solvers and Philip Gill for his careful reading of the paper and helpful suggestions.

References

[1] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, L. C. McInnes, and B. F. Smith, *PETSc home page*. http://www.mcs.anl.gov/petsc, 2001.

[2] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Boston, 1997, Birkhauser Press, pp. 163–202.

[3] ——, *PETSc users manual*, Technical Report ANL-95/11 - Revision 2.1.0, Argonne National Laboratory, 2001.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, first ed., 1984.

[5] C. J. C. Burges, *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery, 2 (1998), pp. 121–167.

[6] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, 2000.

[7] M. C. Ferris and T. S. Munson, *Interior point methods for massive support vector machines*, SIAM Journal on Optimization, 13 (2003), pp. 783–804.

[8] ——, *Semismooth support vector machines*, Mathematical Programming, 101 (2004), pp. 185–204.

[9] R. Fletcher, *Practical Methods of Optimization*, Volume 2: Constrained Optimization, John Wiley and Sons, Chichester and New York, 1981.

[10] E. M. Gertz and S. J. Wright, *OOQP user guide*, Technical Memorandum ANL/MCS-TM-252, Argonne National Laboratory, Argonne, Il, 2001.

[11] ——, *Object oriented software for quadratic programming*, ACM Transactions on Mathematical Software (TOMS), 29 (2003), pp. 49–94.

[12] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, third ed., 1996. ISBN 0-8018-5414-8.

[13] J. Gondzio, *Multiple centrality corrections in a primal-dual method for linear programming*, Computational Optimization and Applications, 6 (1996), pp. 137–156.

[14] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Parallel Computing, 22 (1996), pp. 789–828.

[15] W. D. Gropp and E. Lusk, *User's Guide for* `mpich`*, a Portable Implementation of MPI*, Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[16] T. Joachims, *Making large-scale support vector machine learning practical*, in Advances in Kernel Methods – Support Vector Learning, B. Schölkopf, C. Burges, and A. Smola, eds., MIT Press, 1998, pp. 169–184.

[17] T. Kudo and Y. Matsumoto, *Fast methods for kernel-based text analysis*, in ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, Morristown, NJ, USA, 2003, Association for Computational Linguistics, pp. 24–31.

[18] O. L. Mangasarian and D. R. Musicant, *Lagrangian support vector machines*, Journal of Machine Learning Research, 1 (2001), pp. 161–177.

[19] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization, 2 (1992), pp. 575–601.

[20] E. Osuna, R. Freund, and F. Girosi, *Improved training algorithm for support vector machines*, in Proc. IEEE Workshop on Neural Networks for Signal Processing, 1997, pp. 276–285.

[21] J. Phillips, J. Afonso, A. Oliveira, and L. M. Silveira, *Analog macromodeling using kernel methods*, in ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design, Washington, DC, USA, 2003, IEEE Computer Society, p. 446.

[22] J. Platt, *Fast training of support vector machines using sequential minimal optimization*, in Advances in Kernel Methods – Support Vector Learning, B. Schölkopf, C. Burges, and A. Smola, eds., MIT Press, 1998, pp. 41–65.

[23] ——, *Sequential minimal optimization: A fast algorithm for training support vector machine*, Technical Report TR-98-14, Microsoft Research, 1998.

[24] C. B. S. Hettich and C. Merz, *UCI repository of machine learning databases*, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[25] A. J. Smola and B. Schölkopf, *Sparse greedy matrix approximation in machine learning*, in Proceedings of the 17th International Conference on Machine Learning, Stanford University, CA, 2000, Morgan Kaufmann Publishers, pp. 911–918.

[26] C. B. S.S. Keerthi, S.K. Shevade and K. Murthy, *Improvements to Platt's SMO algorithm for SVM classifier design*, Neural Computation, 13 (2001), pp. 637–649.

[27] I. Steinwart, *On the influence of the kernel on the consistency of support vector machines*, Journal of Machine Learning Research, 2 (2002), pp. 67–93.

[28] V. N. Vapnik, *The nature of statistical learning theory*, Springer Verlag, Heidelberg, DE, 1995.

[29] S. J. Wright, *Primal–Dual Interior–Point Methods*, SIAM Publications, SIAM, Philadelphia, PA, USA, 1996.